Theoretical Computer Science 410 (2009) 5112–5121

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Periodic scheduling with obligatory vacations☆

Jiří Sgall [a], Hadas Shachnai [b,*], Tami Tamir [c]

[a] Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, CZ-11800 Praha 1, Czech Republic

[b] Computer Science Department, The Technion, Haifa 32000, Israel

[c] School of Computer Science, The Interdisciplinary Center, Herzliya, Israel

ARTICLE INFO

Article history:
Received 13 March 2008
Received in revised form 1 July 2009
Accepted 10 August 2009
Communicated by A. Fiat

Keywords:
Periodic scheduling
Vacations
Approximation algorithms

ABSTRACT

We consider a problem of *repeatedly* scheduling n jobs on m parallel machines. Each job is associated with a profit that is gained each time the job is completed, and the goal is to maximize the average profit per time unit. Once the processing of a job is completed, it goes on *vacation* and returns to the system, ready to be processed again, only after its vacation is over. This problem has many applications: in production planning, machine maintenance, video-on-demand and database query processing, among others.

We show that the problem is NP-hard already for jobs with unit processing times and unit profits, and develop approximation algorithms as well as optimal algorithms for certain subclasses of instances. We first show that a 5/3-approximation algorithm can be obtained for instances with unit processing times, using known results for windows scheduling. We then present a preemptive greedy algorithm, which yields a ratio of $e/(e-1)$ to the optimal for general instances, with arbitrary processing times and arbitrary profits. For the special case where all jobs have the same (unit) processing times and the same (unit) profits, we present a 1.39-approximation algorithm. For this case we also show that, when the load generated by an instance is sufficiently large (in terms of n and m), any algorithm which uses no intended idle times yields an optimal schedule.

1. Introduction

We consider a scheduling problem in which jobs need to be scheduled repeatedly. The input is a set of m identical parallel machines and a set of n jobs, $J = \{1, \ldots, n\}$, that need to be scheduled on the machines. All the jobs are ready at time $t = 0$. Each job j is associated with a processing time $p_j \geq 1$, and a window $a_j \geq p_j$; once the processing of j is completed, it goes on *vacation* and returns after $a_j - p_j$ time units, ready to be processed again. Thus, a feasible schedule is one where each machine processes at most one job at any time, and there is a gap of at least $a_j - p_j$ time units between two consecutive executions of job j. The jobs are sequential, i.e., no job can be scheduled on more than one machine at the same time. There is also a profit, b_j, associated with each execution of job j; the goal is to find a feasible schedule that maximizes the average profit per time unit. We call this problem *scheduling with vacations (SWV)*. Our problem arises in many practical scenarios where tasks need to be accomplished periodically, but due to setup times or maintenance requirements there must be some gap between two consecutive executions of a task. The following are two of the applications that motivate our study.

Surveillance camera scheduling: Consider a system of robots carrying surveillance cameras, which patrol an area periodically [16]. Each robot has a predefined path that it needs to patrol, while recording all the events along this path. Upon completion of each patrol, the robot returns to the controller, where the recorded data is downloaded/processed, and the robot is prepared for its next tour. Each patrol j is associated with a revenue b_j, that is gained once the corresponding

☆ A preliminary version of this paper appeared in the Proceedings of the 13th Annual European Symposium on Algorithms (ESA), Palma de Mallorca, October 2005.

* Corresponding author.
E-mail addresses: sgall@kam.mff.cuni.cz (J. Sgall), hadas@cs.technion.ac.il (H. Shachnai), tami@idc.ac.il (T. Tamir).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.08.008

robot completes the tour. The controller can handle at most m robots simultaneously, and it takes p_j time units to complete the processing of robot j. The time it takes robot j to traverse its path is $t_j \geq 1$. The goal of the controller is to process the robots in a way that maximizes the profit gained from all robots, throughout the operation of the system. This yields an instance of SWV, where job j has a processing time p_j, and a window $a_j = p_j + t_j$.

Commercial broadcast: In a broadcasting system which transmits data (e.g., commercials on a running banner) there is some profit associated with the transmission of each commercial. This profit is gained only if some predefined period of time has elapsed since the previous transmission. The goal is to broadcast the commercials in a way that maximizes the overall profit of the system. Thus, we get an instance of SWV, where the window of each job corresponds to the time interval between consecutive transmissions of each commercial.

1.1. Our results and related work

The problem of scheduling with vacations belongs to the class of *periodic scheduling* problems, where each job may be scheduled an infinite number of times. Periodic scheduling is a well-studied problem. Problems of this type arise in many areas, including production planning, machine maintenance, telecommunication systems, video-on-demand, image and speech processing, robot control/navigation systems, and database query processing. Liu and Layland introduced in [15] the periodic scheduling problem: they considered the goal of scheduling each of the jobs such that the average gap between two executions of job j is a_j, using the minimum number of machines. Another early example is the *chairman assignment* problem [17], in which the number of executions of job j in any prefix of the schedule of length t must be at least $\lfloor t/a_j \rfloor$ and at most $\lceil t/a_j \rceil$. For both problems the *earliest deadline first* algorithm was shown to be optimal. A more restricted form of periodic scheduling, known as *pinwheel scheduling*, requires that job j is scheduled at least once in any subsequence of a_j jobs $\{J_{\ell+1}, \ldots, J_{\ell+a_j}\}$ (see, e.g., [12]).

Our problem differs from previously studied variants of periodic scheduling in two major aspects. First, our goal is to maximize the total profit of the server. This may result in a lack of fairness. Indeed, in a schedule which maximizes the profit, some jobs may be waiting indefinitely, while other (more profitable) jobs are repeatedly processed as soon as they return from vacation. In other works (e.g., [15,9,10,12]), the performance of a periodic schedule is measured by some fairness criterion, or (e.g. [3,11]) the two objectives are combined; that is, each job comprises a mandatory and an optional part, with which a non-decreasing reward function is associated. In other variants of the periodic scheduling problem (see, e.g., [1]), the processing of each job incurs a service cost, depending on the time that has elapsed since the last service of this job, and the goal is to minimize the cost of the schedule; however, the cost increases with job delays, implying that an optimal solution requires some fairness. Other periodic scheduling problems arise in the Broadcast Disks environment (e.g., [2]), where satellites broadcast popular information pages to clients, and in TeleText (e.g., [4]), where running banners appear in some television networks.

The other major difference from well-studied variants of periodic scheduling is that, in our problem, due to the vacation requirement, jobs cannot be processed too often. In the *windows scheduling* problem [6,7], the parameter associated with each job gives the *maximal* gap between any two executions of job j; however, it is feasible to schedule a job more often. In the *periodic machine scheduling problem (PMSP)* [5], this parameter specifies the *exact* gap between any two executions of j. To the best of our knowledge, there is no earlier work on periodic job scheduling with a *minimum gap* requirement between consecutive executions of a job.

We first show that SWV is NP-hard already for jobs with unit length and unit profits. Since the total number of different configurations of the system is finite, an optimal solution can be found (inefficiently) by applying the *buffer scheme* designed for the windows scheduling problem [8].

We present several approximation algorithms for various subclasses of instances. We show that a 1.67-approximation algorithm can be obtained for instances with unit processing times, using known results for windows scheduling. For the special case where all jobs have the same (unit) processing times and the same (unit) profits, we present an algorithm with approximation ratio $2 \ln 2 (\approx 1.39)$. For this case we also show that, when the load generated by an instance is sufficiently large (in terms of n and m), any algorithm which uses no intended idle times yields an optimal schedule. Our approximation algorithms for unit length jobs apply a transformation of the instance to an *aligned instance*, where all window sizes satisfy certain properties. Our approximation technique, based on finding an optimal schedule for the resulting aligned instance, builds on a technique of [7] for the windows scheduling problem. To obtain a better approximation ratio, we extend this technique. Our algorithm for unit processing times and unit profits applies the *best align* technique, which finds an aligned instance that yields the best approximation ratio for the original instance.

We then present a preemptive greedy algorithm, which yields a ratio of $e/(e-1) \approx 1.58$ to the optimal for general instances in a preemptive model, with arbitrary processing times and arbitrary profits.

2. Preliminaries

2.1. Definitions and notation

As mentioned above, each job j is associated with three parameters: a_j, b_j, and p_j, denoting the *scheduling window* (for short, *window*), the *profit*, and the *processing time* of job j, respectively; we assume that these parameters are positive

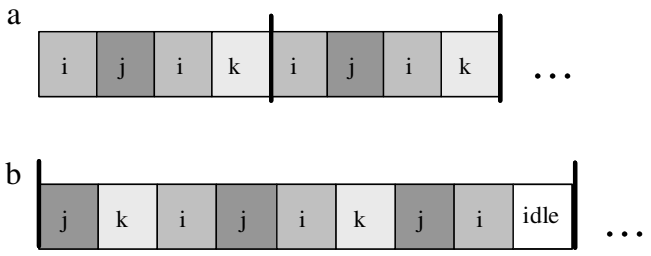

Fig. 1. Two possible periodic schedules of the same instance.

integers. The *load* incurred by job j is defined as $\ell(j) = p_j/a_j$. The best service job j can receive is to be processed for the first time as early as possible, and then whenever it returns from vacation (i.e., no waiting time). Thus, the load of job j represents the average processing requirement of j per time unit. The total load of the input is $L(J) = \sum_{j\in J} \ell(j)$. In the special case of unit processing times (i.e., for all j, $p_j = 1$), we get that $\ell(j) = 1/a_j$, and $L(J) = \sum_{j\in J} 1/a_j$. Each job is associated with an additional parameter, b_j, the profit gained from each execution of job j.

We say that a job j is *denser* than j' if $b_j/p_j > b_{j'}/p_{j'}$, or $b_j/p_j = b_{j'}/p_{j'}$ and $j > j'$. A denser job achieves larger profit per unit time of its execution; the second part of the definition only breaks ties consistently for the whole instance. Let $J_{\geq j}$ denote the set of jobs at least as dense as j.

To define the profit of a schedule formally, let $compl_T^S(j)$ denote the number of times job j is completed in the first T time units in schedule S. The profit of schedule S is defined as

$$profit(S) = \liminf_{T\to\infty} \frac{1}{T} \sum_{j=1}^{n} compl_T^S(j) b_j. \tag{1}$$

Note that in calculating the profit of a schedule S we consider an infinite time horizon; thus, $profit(S)$ is not affected by changes in a finite part of the schedule.

A schedule is periodic with period q if, for any T, the jobs scheduled at time T are the same as the jobs scheduled at time $T + q$. From the definition of $profit(S)$ it is not clear that there exists an optimal schedule. However, as we show in Theorem 2.2, for any instance there exists an optimal schedule which is also periodic.

'For a schedule S with period q, 'lim inf' in (1) can be replaced by 'lim' which always exists, and it is actually equal to the value of the remaining expression for $T = q$. For a periodic schedule, it is meaningful to consider the load (or, relative frequency) of a job j in S, denoted by $\ell^S(j)$: this is the number of time slots in a period in which j is scheduled, divided by q. Let $L^S(J') = \sum_{j\in J'} \ell^S(j)$ denote the load of a set of jobs $J' \subseteq J$ in a schedule S.

Example. Fig. 1 presents two possible schedules on a single machine of an instance consisting of three unit-length jobs (i, j, k) having windows $(2, 3, 4)$ (that is, vacations $(1, 2, 3)$) and profits $(1, 5, 1)$, respectively. Schedule (a) has period 4 and profit (=average profit per time slot) $(1 + 5 + 1 + 1)/4 = 2$. Schedule (b) has period 9 and profit $(5 \cdot 1 + 3 \cdot 5)/9 = 2.22$. Note that in the last slot of the period the machine is idle. This idle is inevitable because all the jobs are on vacation during this slot. A simple upper bound for the profit from this instance is $5/3 + 1/2 + 1/4 = 2.42$. For schedule (a), we have $\ell^S(i) = 2/4$, $\ell^S(j) = \ell^S(k) = 1/4$, and taking $J' = \{i, j, k\}$, $L^S(J') = 1$; for schedule (b) we have $\ell^S(i) = 3/9$, $\ell^S(j) = 3/9$, $\ell^S(k) = 2/9$, and $L^S(J') = 8/9$.

2.2. NP-hardness

The hardness of SWV with a single machine, unit profits, and unit processing times is shown by a reduction from the *periodic machine scheduling problem (PMSP)*, which is known to be NP-hard (see in [5]). In the PMSP, there is a set of n devices; the j-th device requires maintenance every a_j time units, where $\sum_{j=1}^n 1/a_j \leq 1$. The maintenance of any device takes one time unit, and at any time only one device can be attended. It is assumed that the first maintenance of device j can be done in any of the first a_j time units. The goal is to find a maintenance schedule that satisfies *exactly* all the requirements; that is, for all j, device j is maintained exactly once in any window of a_j time units.

Theorem 2.1. *SWV is NP-hard, even with a single machine, unit profits and unit processing times.*

Proof. Given an instance of the PMSP with n devices in which the j-th device requires maintenance every a_j time units, construct an instance of SWV with unit profits, unit processing times, and scheduling windows a_j for jobs $j = 1, \ldots, n$. Since $\sum_{j=1}^n 1/a_j \leq 1$, there is a feasible schedule for the PMSP if and only if the average profit per time unit from job j is exactly $1/a_j$; that is, if and only if there is a schedule of the SWV instance with average profit $\sum_{j=1}^n 1/a_j$. Note that $\sum_{j=1}^n 1/a_j$ is the maximal possible profit for this SWV instance, and it is achieved only when each job is processed immediately after it returns from vacation. This implies that the transformation above reduces the NP-hard PMSP to SWV. ■

The SWV instances produced in the above reduction have total load at most 1. In Section 4.2, we characterize several classes of instances with $L(J) > 1$ for which the SWV problem is solvable in polynomial time.

2.3. Optimal periodic schedules and an exponential algorithm

We first describe an exponential time algorithm that finds an optimal schedule among the periodic ones. Then, based on some insights from this algorithm, we observe that a non-periodic schedule cannot improve upon the best periodic schedule.

The general buffer scheme [8] is a tool designed for solving the windows scheduling problem optimally. We explain below how it can be modified to solve the problem of scheduling with vacations optimally. We consider unit-length jobs and arbitrary profits; however, the algorithm can be easily extended to handle jobs with arbitrary lengths (both preemptive and non-preemptive) similarly to the extension for windows scheduling shown in [8]. The buffer scheme is based on representing the system as a non-deterministic finite state machine, in which any directed closed walk corresponds to a valid periodic schedule and vice versa.

Let $a^* = \max_j\{a_j\}$. The state of a schedule is represented using a set of buffers, $B_0, B_1, \ldots, B_{a^*-1}$. Each job is stored in some buffer. A job is stored in B_i when i time slots remain till the end of its vacation. Initially, all jobs are ready to be processed so they are all stored in B_0. In each iteration, the scheme schedules at most m jobs from B_0 and updates the content of the buffers:

- For all $i > 0$, all jobs stored in B_i are moved to B_{i-1} (note that none of them is scheduled).
- Each scheduled job j is moved from B_0 to B_{a_j-1} (this ensures that at least $a_j - 1$ time slots will elapse before j is scheduled again).

Note that the total number of buffer configurations is at most $\Pi_j a_j$. In the state machine, each configuration is represented by a vertex, and there is an edge connecting configuration f to configuration g if it is possible to move from the state corresponding to f to the state corresponding to g in a single time slot. Each edge $e = (f, g)$ has a weight b_e whose value is the profit gained by moving from f to g; this is the total profit of the jobs selected for execution.

A periodic schedule with period q corresponds to a closed walk of length q in the state machine. The profit of this schedule is given by the average edge weight along this walk. Note the subtle difference between a walk and a schedule. Since the walk may not contain the initial configuration, during the first q steps of the schedule, the actual configurations may differ from those in the corresponding vertices of the walk. However, if we take a walk, it still generates a feasible schedule, and the correspondence is one-to-one.

To find an optimal periodic schedule, notice that each closed walk can be decomposed into simple cycles. By averaging, the average profit of the best cycle is at most that of the best walk. Thus a cycle with maximal mean weight corresponds to an optimal schedule. Such a cycle can be found by using the classic *Max Cycle Mean* algorithm of Karp [13]. The running time is polynomial in the graph size, which is polynomial in $\Pi_j a_j$.

Clearly, this algorithm is applicable only for small instances; however, since the problem is NP-hard, it is unlikely to admit a polynomial time algorithm for general instances.

Theorem 2.2. *Any instance J has an optimal schedule which is periodic.*

Proof. Let X be the profit of the best periodic schedule obtained by the algorithm above. For a contradiction, assume that, for some $\varepsilon > 0$, there exists a schedule S with profit at least $X + 2\varepsilon$.

Define a configuration at time T as in the previous algorithm. Since the number of configurations is finite, and the schedule is infinite, there must exist a configuration which appears in S infinitely often at times $T_1, T_2, \ldots$. Since the profit of S is strictly larger than $X+\varepsilon$, one of the time intervals between T_i and T_{i+1} has profit at least $X+\varepsilon$. (In fact, there are infinitely many such time intervals.) By repeating the scheduling section between T_i and T_{i+1} we get a periodic schedule with the same profit, which is a contradiction to the assumption that X is the maximal profit of a periodic schedule. ■

In the rest of the paper we consider only periodic schedules.

2.4. A technical lemma

The next lemma is used to compare our schedules to an optimal schedule S^*. Essentially it asserts that, to be R-competitive, it is sufficient to guarantee that, in our schedule, for any threshold, the load of jobs denser than the threshold is at least $1/R$ of their load in some optimal schedule. In fact, it is sufficient to have this condition hold on the average over several schedules.

Lemma 2.3. *Let S^* be an arbitrary schedule for an instance J and let $R \geq 1$.*

(i) *Suppose that, for a schedule S of J, and for any job j,*

$$L^S(J_{\geq j}) \geq L^{S^*}(J_{\geq j})/R, \tag{2}$$

then $\mathit{profit}(S) \geq \mathit{profit}(S^*)/R$.

(ii) *Let $\mathcal{S}$ be some set of feasible schedules of J, and suppose that the schedule S is selected by some probability distribution over $\mathcal{S}$. Also, assume that $\mathbf{Exp}_{S\in\mathcal{S}}[L^S(J_{\geq j})] \geq L^{S^*}(J_{\geq j})/R$. Then one of the schedules $S \in \mathcal{S}$ satisfies $\mathit{profit}(S) \geq \mathit{profit}(S^*)/R$.*

Proof. Recall that the profit of a schedule is the average profit per time slot, and let $J_\beta = \{j \in J \mid b_j \geq \beta\}$. To show (i) we note that, for any schedule S,

$$\mathit{profit}(S) = \int_{\beta\geq 0} L^S(J_\beta)\,\mathrm{d}\beta.$$

To see this, note that each job j contributes to the integral exactly for $\beta \in [0, b_j]$, and the contribution is equal to $\ell^S(j)$. Now, since (2) holds for all j, it follows that, for any $\beta > 0$, the set of jobs J_β satisfies $L^S(J_\beta) \geq L^{S^*}(J_\beta)/R$. Therefore, $profit(S) \geq profit(S^*)/R$. The proof of (ii) is similar, since the inequality is linear and thus holds also in expectation. ■

3. Approximating by aligned instances

In this section we study a natural approach for handling instances with unit processing times, inspired by previous work on *window scheduling* (see, e.g., [7]). We call an instance *aligned* if there exists an integer q such that for each j, $a_j = q \cdot 2^{\alpha_j}$ for some integer $\alpha_j \geq 0$, or $a_j = 1$. (Note that this generalizes the case where all the windows are powers of 2.) The following result is due to [7].

Theorem 3.1. *An optimal solution for an aligned instance can be computed in polynomial time.*

For completeness, we describe such an optimal solution.

Schedule the jobs in order of increasing windows, always keeping the period of the schedule equal to the maximal window processed so far. The machines are utilized in an arbitrary order. When a new job is processed, if needed, increase the period by doubling the period and repeating the current schedule, until the period is equal to the currently processed window. Then, if possible, schedule the current job in an empty slot on some machine. Otherwise, if the current job has a bigger profit than some of the scheduled jobs, run this job in a slot of the scheduled job with the smallest profit. (This job can still remain scheduled in some other slots, due to a possible previous doubling of the period.)

An alternative view of this schedule S will be useful. Let j be the densest job such that $L(J_{\geq j}) \geq m$. Then the schedule above has the property that all jobs j' denser than j are scheduled as often as possible, i.e., $\ell^S(j') = \ell(j')$, and no jobs lighter than j occur in the schedule. The borderline job j is scheduled in all the remaining slots.

Note that the solution described above works also if we start not with an empty schedule but with some schedule with period q, where some slots are already used by other jobs, assuming that the jobs are sufficiently dense (e.g., in the case of unit jobs and unit profits, where we use this observation).

It follows that a simple 2-approximation algorithm can be achieved by rounding the window of each job to the next higher power of 2 and using the optimal algorithm for aligned instances. (And this is no better than 2-approximation: Consider an instance with a single job with $a_j = 2^\alpha + 1$ for a large α.)

Below we give an algorithm that refines this idea and achieves a better approximation ratio. This algorithm is deterministic: Randomization is used only in the analysis.

3.1. A 5/3-approximation algorithm

Algorithm SimpleAlign
Produce instance J' where any window $a_j > 1$ is rounded up to the next power of 2.
Produce instance J'' where any window $a_j > 1$ is rounded up to the next number of the form $3 \cdot 2^\alpha$, for some integer $\alpha \geq 0$.
Find an optimal schedule S' for J'.
Find an optimal schedule S'' for J''.
Choose the better of S' and S''.

Theorem 3.2. SimpleAlign *is a* $5/3 \approx 1.67$ *approximation algorithm for unit processing times and arbitrary profits.*

Proof. Using Theorem 3.1, SimpleAlign runs in polynomial time.

Consider a distribution which chooses S' with probability $2/5$ and S'' with probability $3/5$. We prove that the condition of Lemma 2.3 is satisfied with $R = 5/3$. We use the following claim, where by $\ell^{J'}(j)$ we denote the load of a job j rounded as in the instance J', and similarly for $\ell^{J''}$ and J''.

Claim 3.3. *For any job* j, $\frac{2}{5}\ell^{J'}(j) + \frac{3}{5}\ell^{J''}(j) \geq \ell(j)/R$.

Proof. We consider two cases:

(i) If for some integer α, $2^{\alpha+1} \leq a_j \leq 3 \cdot 2^\alpha$, then

$$\frac{2}{5}\ell^{J'}(j) + \frac{3}{5}\ell^{J''}(j) \geq \frac{2}{5} \cdot \frac{1}{2^{\alpha+2}} + \frac{3}{5} \cdot \frac{1}{3 \cdot 2^\alpha} = \frac{3}{5 \cdot 2^{\alpha+1}} \geq \frac{\ell(j)}{R}.$$

(ii) Otherwise, for some integer α, $3 \cdot 2^\alpha < a_j < 2^{\alpha+2}$, in which case

$$\frac{2}{5}\ell^{J'}(j) + \frac{3}{5}\ell^{J''}(j) \geq \frac{2}{5} \cdot \frac{1}{2^{\alpha+2}} + \frac{3}{5} \cdot \frac{1}{3 \cdot 2^{\alpha+1}} = \frac{1}{5 \cdot 2^\alpha} \geq \frac{\ell(j)}{R}.$$ ■

Now, given an optimal schedule S^* and a job j, we prove that Lemma 2.3 can be applied here. We distinguish between two cases.

(a) First, assume that both $L^{S'}(J_{\geq j}) < m$ and $L^{S''}(J_{\geq j}) < m$. Then $L^{S'}(J_{\geq j}) = \sum_{j \in J_{\geq j}} \ell^{J'}(j)$, by the construction of an optimal schedule for aligned instances, and similarly for S''. From Claim 3.3, we obtain $\mathbf{Exp}[L^S(J_{\geq j})] \geq L(J_{\geq j})/R \geq L^{S^*}(J_{\geq j})/R$.

(b) Otherwise it must be the case that $L^{S'}(J_{\geq j}) = m$ or $L^{S''}(J_{\geq j}) = m$ (note that the load in a schedule cannot be larger than m). Either way, this implies that $L(J_{\geq j}) \geq m$ in the original instance. In both J' and J'', the size of each window is at most doubled; therefore $L^{S'}(J_{\geq j}) \geq m/2$ and $L^{S''}(J_{\geq j}) \geq m/2$. Then the average load is bounded by $\mathbf{Exp}[L^S(J_{\geq j})] \geq 2/5 \cdot m + 3/5 \cdot m/2 = 7m/10 \geq m/R \geq L^{S^*}(J_{\geq j})/R$.

The proof is completed by an application of Lemma 2.3. ■

We remark that our analysis of SimpleAlign is tight. Suppose that, for a large α, we have two jobs with windows $2 \cdot 2^\alpha + 1$ and $3 \cdot 2^\alpha + 1$, with unit weight. In the optimal schedule, the load is close to $5/(6 \cdot 2^\alpha)$. In J', the rounded windows are both $4 \cdot 2^\alpha$ and the load in S' is $1/(2 \cdot 2^\alpha)$. In J'', the rounded windows are $3 \cdot 2^\alpha$ and $6 \cdot 2^\alpha$; thus the load in S'' is also $1/(2 \cdot 2^\alpha)$. The ratio for a large α approaches $(5/6)/(1/2) = 5/3$.

The following theorem shows an interesting consequence for instances with unit profit, where we can compare against the load of the instance instead of the optimal schedule.

Theorem 3.4. *For any instance J with unit profits, the load achieved by* SimpleAlign *is at least $0.6 \cdot L(J)$. In particular, if $L(J) \geq 5m/3$, it produces the optimal schedule with a full profit.*

Proof. If $L^{S'}(J) = m$ or $L^{S''}(J) = m$, we are done. Otherwise $L^{S'}(J) = L(J')$ and $L^{S''}(J) = L(J'')$. By the construction of an optimal schedule for aligned instances, and from Claim 3.3, summing over all jobs, we obtain $\mathbf{Exp}[L^S(J)] \geq L(J)/R = 0.6 \cdot L(J)$. ■

4. Unit processing times and unit profits

In this section we consider the special case where all jobs have the same (unit) processing times and the same (unit) profit. Thus, the average profit is equal to the load of the schedule. In particular, if the schedule has no idle time, it is optimal. By Theorem 2.1, the problem is NP-hard already for this case, even on a single machine.

4.1. A 1.39-approximation algorithm

As in Section 3, we use in our algorithm *aligned instances*; however, instead of using two schedules with periods 2 and 3 times a power of 2, the next algorithm uses one of k schedules with periods $q = k+1, k+2, \ldots, 2k$ times a power of 2. For a large but constant k, the approximation ratio approaches $2 \ln 2 \approx 1.39$. The jobs with large windows are rounded similarly to the rounding in Section 3.1.

The idea of using more than two schedules as in Section 3.1 faces several difficulties. First, we cannot use the same argument as in Theorem 3.2, case (b) in the proof. When we have more rounded instances and only one of them has a full load of big jobs, the obtained ratio is not sufficiently good. This is the reason why we can analyze the algorithm in this section only for jobs with unit profit.

Another obstacle is that we need to handle the jobs with windows smaller than k. These jobs are handled separately. Given a constant q and a set of jobs with windows at most q, we find an optimal schedule with period q in polynomial time. If the number of machines is constant, then the number of such schedules is constant and we can find an optimal one simply by exhaustive search.

For arbitrary m, we can use, for example, Lenstra's polynomial algorithm for integer programming in fixed dimension (see [14]), similarly to its other applications in scheduling. Since q is a constant, we have a constant number of job types specified by their windows, and for each type of job we have a constant number of patterns specifying in which time slots it runs. The variables of the integer program correspond to the number of jobs of each type following each pattern; the number of these variables is a constant exponential in q. The constraints specify that (i) the number of scheduled jobs of each type equals the number of such jobs in the instance, and that (ii) the number of jobs scheduled in any given time unit is at most m.

Formally, we number the patterns by $1, \ldots, Q$, where $Q \leq 2^q$. Let P_i denote the set of slots associated with pattern i, and $r_i = |P_i|$ be the profit of pattern i. Also, denote by $Invalid_j$ the set of patterns which are invalid for type j (due to vacation constraints), and let x_{ij} denote the number of jobs of type j that follow pattern i. Finally, n_j is the number of jobs of type j, $1 \leq j \leq q$. Then the integer program can be formulated as follows.

$$
\begin{array}{llll}
(IP) & \max & \sum_i r_i \sum_j x_{ij} & \\
 & s.t. & \sum_i x_{ij} \geq n_j & \text{for } j = 1, \ldots, q \\
 & & \sum_j \sum_{\{i \mid \ell \in P_i\}} x_{ij} \leq m & \text{for } \ell = 1, \ldots, q \\
 & & x_{ij} = 0 & \text{for all } j,\ i \in Invalid_j \\
 & & x_{ij} \in \{0, 1, \ldots, n_j\} & \text{for all } j,\ i \notin Invalid_j.
\end{array}
$$

Feasible integral solutions then correspond to schedules in a straightforward way. Given an instance, the integer program can be produced in linear time, and solved in time polylogarithmic in n and m, with a multiplicative constant doubly exponential in q. This follows from the fact that the time complexity of Lenstra's algorithm is exponential in the dimension of the program, but polynomial in the logarithms of the coefficients. In the above IP, the dimension is $O(2^q)$ and the coefficients are bounded by $\max\{n, m\}$.[1]

[1] In fact, among the jobs with windows at most q, the algorithm schedules only those whose windows are at most $k/4$.

Algorithm BestAlign
Let $\varepsilon > 0$ be given.
Let $K = \lceil 1/\varepsilon \rceil$.
For all $k \in \{K, 4K, \ldots, 4^x K, \ldots, 4^K K\}$ do
 For all $q = \{k+1, k+2, \ldots, 2k\}$ do
  Let J' be the set of jobs with windows at most $k/4$.
  Let J'' be the set of jobs with windows larger than k.
  Generate from J'' the aligned instance J''' by rounding up the window of each job to the next number of the form $q2^\alpha$, for some integer $\alpha \geq 0$.
  Find the best schedule with period q for J'.
  Schedule J''' in the remaining slots, using an optimal algorithm for aligned instances.
Output the best schedule over all values of k and q.

Theorem 4.1. BestAlign *is a* $(2 \ln 2 + O(\varepsilon))$*-approximation algorithm for the jobs with unit processing times and unit profits. In other words, there exists an R-approximation algorithm for any* $R > 2 \ln 2 \approx 1.39$.

Proof. We first note that BestAlign can be implemented in polynomial time. Indeed, by the above discussion, we can find in polynomial time an optimal schedule for J', since q is a constant depending on ε. Also, we can schedule the aligned instance J'' optimally, using Theorem 3.1.

Let S^* be an optimal schedule for a given instance. For the proof, fix a value of k among those used in the algorithm, so that the contribution of the jobs with windows between $k/4$ and k to $L(S^*)$ is at most $\varepsilon \cdot L(S^*)$. Since there are $\lceil 1/\varepsilon \rceil + 1$ choices of k, one of them has a sufficiently small contribution. This defines the sets of jobs J' and J''. Let S' and S'' denote the schedule S^* restricted to J' and J'', respectively, i.e., all the other jobs are simply removed from the schedule.

For any q, let S_q be the schedule generated by BestAlign for this choice of q and for k fixed as above. If any of S_q has load m, it is optimal for J and the theorem follows. Thus, for the remaining proof we can assume that S_q always schedules all the jobs in J''' with their maximal load, i.e., $L^{S_q}(J''') = L(J''')$.

We prove that one of the schedules S_q has load at least $3/4 \cdot L^{S'}(J') + L(J'')/(2 \ln 2 + O(\varepsilon))$. The theorem then follows since, by the choice of k, $L^{S^*}(J) \leq (1 + O(\varepsilon))(L^{S'}(J') + L^{S''}(J''))$; furthermore, $L^{S''}(J'') \leq L(J'')$, and $2 \ln 2 > 4/3$.

The schedule generated by BestAlign for J' has load at least $3/4$ of $L(S')$. Indeed, if we take in S' a segment of length $q - k/4$ with the maximal load and append to it $k/4$ empty slots, we get a schedule with period q and load at least $3/4$ of $L(S')$. The optimal schedule for J' with period q chosen by BestAlign has at least the same load.

Now we complete the proof of the theorem by showing that, for some distribution over the schedules S_q, $\mathbf{Exp}[L^{S_q}(J''')] \geq L(J'')/(2 \ln 2 + O(\varepsilon))$. Define the probability distribution so that the probability of choosing S_q is $\pi_q = X/(q-1)$, where X is chosen so that the sum of the probabilities is 1, i.e.,

$$\left(\frac{1}{k} + \cdots + \frac{1}{2k-1}\right) X = 1. \tag{3}$$

We proceed with the proof job by job, as in Theorem 3.2. Since S_q schedules all the jobs with maximal rate, it is sufficient to prove for each job j that its expected load in J''' (i.e., after rounding) is at least $\ell(j)/(2 \ln 2 + O(\varepsilon))$. Assume that the integers $\alpha \geq 0$ and $t \in \{k, \ldots, 2k-1\}$ satisfy $t2^\alpha < a_j \leq (t+1)2^\alpha$ (for each j, there exist unique values of t and α). Then the average load of j in J''' is

$$\begin{aligned}\sum_{q=k+1}^{t} \pi_q \cdot \frac{1}{q2^{\alpha+1}} + \sum_{q=t+1}^{2k} \pi_q \cdot \frac{1}{q2^\alpha} &= t\left(\sum_{q=k+1}^{t} \pi_q \cdot \frac{1}{2q} + \sum_{q=t+1}^{2k} \pi_q \cdot \frac{1}{q}\right)\frac{1}{t2^\alpha} \\ &\geq t\left(\sum_{q=k+1}^{t} \pi_q \cdot \frac{1}{2q} + \sum_{q=t+1}^{2k} \pi_q \cdot \frac{1}{q}\right)\ell(j).\end{aligned}$$

It remains to bound the coefficient on the right-hand side. By the definition of π_q, we have

$$\frac{\pi_q}{q} = \frac{X}{(q-1)q} = \left(\frac{1}{q-1} - \frac{1}{q}\right) X. \tag{4}$$

Using (4) and telescoping the sums, we have

$$t\left(\sum_{q=k+1}^{t} \pi_q \cdot \frac{1}{2q} + \sum_{q=t+1}^{2k} \pi_q \cdot \frac{1}{q}\right) = tX\left(\frac{1}{2k} - \frac{1}{2t} + \frac{1}{t} - \frac{1}{2k}\right) = \frac{X}{2}.$$

Using (3), we have

$$\frac{2}{X} = 2\left(\frac{1}{k} + \cdots + \frac{1}{2k-1}\right) \le 2\int_k^{2k} \frac{\mathrm{d}x}{x} + \frac{2}{k} = 2(\ln(2k) - \ln k) + \frac{2}{k} \le 2\ln 2 + O(\varepsilon).$$

This completes the proof of the theorem. ■

By a similar example as for SimpleAlign, we can show that our analysis of BestAlign is tight.

4.2. Instances with large load

In the case where $p_j = b_j = 1$, the goal is to maximize the utilization of the machines. A simple observation is that the best one can expect to achieve is a schedule in which the machines are busy all the time. In this section we focus on two cases where such a schedule can be generated.

Theorem 3.4 shows that, if $L(J) \ge 5m/3$, then SimpleAlign produces a schedule with load m. The next theorem gives a (generally more restrictive) condition under which *any* greedy algorithm is optimal for scheduling jobs with unit lengths and unit profits. In the resulting schedule, we get full utilization of all machines. In other words, no machine is ever idle, regardless of the jobs selected to be scheduled at any time slot.

Theorem 4.2. *Suppose that $n = km+r$ for some $r < m$ (i.e., $k = \lfloor n/m \rfloor$ and $r = n \bmod m$). If $L(J) > mH_k - 1 + (r+1)/(k+1)$, then any algorithm with no intended idle times achieves the optimal profit. In particular, for $m = 1$, the condition is $L(J) > H_{n+1} - 1$.*

Proof. We show that, if a machine is idle, $L(J)$ must be small. Assume that some machine is idle for the first time at t. Thus, none of the n jobs is available, meaning that all jobs are either processed on the other machines or on vacation. Order the jobs by the last execution up to time t, including possible executions on the other $m-1$ machines at time t. The last $m-1$ jobs in this order are possibly executed at time t; we have no bound on their vacation, so each can contribute as much as 1 to $L(J)$. The previous m jobs in this order are scheduled no later than at time $t-1$; they are on vacation at time t, thus they have $a_j \ge 2$ and contribute at most $1/2$ each to $L(J)$. Similarly, the previous m jobs are scheduled no later than at time $t-2$ and contribute at most $1/3$ each to $L(J)$, and so on. The last $r+1$ jobs contribute at most $1/(k+1)$ each. Thus, the total load of the input is

$$L(J) \le m - 1 + m\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k}\right) + \frac{r+1}{k+1} = mH_k - 1 + \frac{r+1}{k+1}.$$

Therefore, $L(J) > mH_k - 1 + (r+1)/(k+1)$ implies that there is no idle time. ■

5. A greedy algorithm for preemptive scheduling

We describe below a simple greedy algorithm which improves upon the algorithm in Section 3.

This algorithm works also for arbitrary-length jobs in the following preemptive model. A job may be preempted at any time and the execution of a preempted job can be resumed at any time; the vacation constraint refers only to the gaps between *different* executions of a job. Thus the job cannot be scheduled during $a_j - p_j$ time steps following its completion. Note that this corresponds to the motivation of surveillance camera scheduling.

We can also consider an alternative preemptive model where the restriction is weaker; namely, any two times when a job j starts differ by at least a_j. The greedy algorithm and our analysis are valid also in this model.

Algorithm Greedy

At any time t, schedule for one time unit the m densest available jobs. (Recall that m is the number of machines.)

It is easy to see that Greedy always generates a periodic schedule, as the possible number of different states is finite.

Theorem 5.1. Greedy *is an $e/(e-1)$-approximation algorithm.*

Proof. Sort the jobs in non-decreasing order of the ratio b_j/p_j. Using this order renumber the jobs such that J_1 is the lightest and J_n is the densest. Let S be the schedule of Greedy; then we define $L^S(J_{>j}) = L^S(J_{\ge j+1})$. We use in the proof the next two claims.

Claim 5.2. *For any job j,* $\dfrac{\ell^S(j)}{\ell(j)} \ge 1 - \dfrac{L^S(J_{>j})}{m}$.

Proof. If the distance between two executions of j in S is $a_j + d$, then dense jobs must occupy d out of $a_j + d$ time slots; note that this holds in both models of preemptive scheduling. Considering the average of all executions of j, we obtain $L^S(J_{>j})/m \ge d/(a_j + d)$. Now, we have that $\ell^S(j)/\ell(j) = a_j/(a_j + d) = 1 - d/(a_j + d)$. This completes the proof. ■

Claim 5.3. *For any job j,* $\dfrac{L^S(J_{\ge j})}{m} \ge 1 - \mathrm{e}^{-L(J_{\ge j})/m}$.

Proof. We prove the claim using backward induction on j. Specifically, we show that, if the claim holds for $J_{>j}$, then it also holds for $J_{\ge j}$.

For the base case we take $j = n$ (i.e., the densest job), then $J_{>j}$ is empty and the claim trivially holds.

Now, assume that the claim holds for $J_{>j}$, and let $x = L(J_{>j})/m, y = L^S(J_{>j})/m$ and $z = \ell(j)/m$. By the induction hypothesis and Claim 5.2, we have

$$\begin{aligned}\frac{L^S(J_{\geq j})}{m} &= \frac{L^S(J_{>j})}{m} + \frac{\ell^S(j)}{m}\\ &\geq y + z(1-y) = y(1-z) + z\\ &\geq (1-\mathrm{e}^{-x})(1-z) + z = 1 - \mathrm{e}^{-x}(1-z)\\ &\geq 1 - \mathrm{e}^{-x}\mathrm{e}^{-z} = 1 - \mathrm{e}^{-L(J_{\geq j})/m}.\end{aligned}$$

The third inequality follows from the fact that $\mathrm{e}^{-z} \geq 1 - z$ for all $z \geq 0$, and the last equality holds since $L(J_{\geq j})/m = x + z$. ■

To complete the proof of the theorem, let $x = L(J_{\geq j})/m$. If $x \geq 1$, then we have $L^S(J)/m \geq 1 - \mathrm{e}^{-1}$, from Claim 5.3. We are done, since the optimum (i.e., the load in S^*) is at most m.

Otherwise, $1 - \mathrm{e}^{-x} \geq (1 - \mathrm{e}^{-1})x$ for $0 \leq x \leq 1$, as the left-hand side is a concave function, the right-hand side is linear and at $x = 0$ and $x = 1$ equality holds. Thus, from Claim 5.3, we have $L^S(J_{\geq j}) \geq (1 - \mathrm{e}^{-1})L(J_{\geq j}) \geq (1 - \mathrm{e}^{-1})L^{S^*}(J_{\geq j})$, and we are done as well.

Finally, using Lemma 2.3, the ratio is at most $1/(1 - \mathrm{e}^{-1}) = e/(e - 1)$. ■

We now show that our analysis of Greedy is tight: In fact, $e/(e-1) \approx 1.58$ is a lower bound for Greedy already for unit-length jobs. Consider the following example. Let a be an integer and let $A = \{a + 1, a + 2, \ldots, \lfloor ea \rfloor\}$. We have $\sum_{i \in A} 1/i = \ln \lfloor ea \rfloor - \ln a + O((ea)^{-2}) \leq \ln e + O((ea)^{-2})$. For a sufficiently large, this sum is arbitrarily close to 1. Let $m = LCM\{a + 1, a + 2, \ldots, \lfloor ea \rfloor\}$. That is, each of the numbers $a + 1, a + 2, \ldots, ea$ divides m. The instance consists of m machines, and m jobs having window $a_j = i$ for each $i = a + 1, a + 2, \ldots, \lfloor ea \rfloor$. The profits are all almost equal to 1, with a slight preference to jobs having long vacation.

The Greedy schedule: In the first slot, the most profitable jobs are those having vacation $\lfloor ea \rfloor$, so these m jobs are scheduled first on each of the m machines. Next, Greedy schedules on all m machines the jobs whose vacation is $\lfloor ea \rfloor - 1$, and so on. The resulting schedule has period $\lfloor ea \rfloor$ repeating on each machine $[\lfloor ea \rfloor, \lfloor ea \rfloor - 1, \ldots, (a + 1), *, \ldots, *]$, where stars denote a consecutive idle slots. The average profit per slot on each of the machines is therefore arbitrarily close to $(ea - a)/(ea) = (e - 1)/e$.

An optimal schedule: For each i, m/i machines schedule with no idle time all the jobs having the window i. Since $\sum_{i \in A} 1/i \approx 1$, there are enough jobs to 'saturate' this way all the machines with no idle times. Therefore the average profit per slot is arbitrarily close to 1 and the approximation ratio tends to $e/(e - 1)$.

6. Discussion and open problems

We introduced a variant of the classic periodic scheduling problem in which, once the processing of a job j is completed, it goes on vacation and returns after $a_j - p_j$ time units, ready to be processed again. We developed algorithms that yield constant factor approximations where the objective is to maximize the total profit of the schedule. Several interesting questions remain open.

- Are there subclasses of instances on which the problem admits polynomial time approximation schemes?
- We expect the Greedy algorithm to perform well in practice. It would be interesting to analyze (via simulation) its average-case performance on randomly generated instances, and compare these results to the average-case performance of BestAlign.
- Our results for arbitrary-length jobs apply only to *preemptive* scheduling. An interesting avenue for further research is to develop approximation algorithms for *non-preemptive* scheduling of the jobs.[2]
- Consider the generalization of our problem where each job j has several possible vacations; whenever j is completed it specifies the duration of its next vacation. This problem has applications, for example, in scheduling commercials on TV. Depending on the slot in the evening program to which a commercial is assigned, the system can schedule it again at some later time. Thus, the vacation of each job can take several possible values. The goal is to schedule the jobs subject to the varying vacation constraints so as to maximize the total profit.
- Finally, we measure the quality of a solution by the profit gained from the scheduled jobs. In many scenarios, there is a *cost* associated with an extended vacation of a job. Formally, each job j has to be processed exactly $a_j - p_j$ time units after completing its previous execution. A cost c_j is incurred for any time unit in which job j is waiting to be processed (once j has returned from vacation). The objective is to find a schedule which minimizes the total cost.

Acknowledgments

We thank Gerhard Woeginger for stimulating discussions on this paper. Also, we thank an anonymous referee for many insightful comments and suggestions. The first author was partially supported by the Institute for Theoretical Computer Science, Prague (project 1M0545 of MŠMT ČR) and grant IAA100190902 of GA AV ČR.

[2] Note that, as shown in Section 2.3, the problem can be solved optimally by the (exponential time) buffer scheme.

References

[1] S. Anily, J. Bramel, Periodic scheduling with service constraints, Operations Research 48 (2000) 635–645.
[2] S. Acharya, M.J. Franklin, S. Zdonik, Dissemination-based data delivery using broadcast disks, IEEE Personal Communications 2 (6) (1995) 50–60.
[3] H. Aydin, R. Melhem, D. Mosse, P. Mejia-Alvarez, Optimal reward-based scheduling of periodic real-time tasks. in: 20th IEEE Real-Time Systems Symp., 1999.
[4] H. Ammar, J.W. Wong, The design of teletext broadcast cycles, Performance Evaluation 5 (4) (1985) 235–242.
[5] A. Bar-Noy, R. Bhatia, J. Naor, B. Schieber, Minimizing service and operation costs of periodic scheduling, Mathematics of Operations Research 27 (3) (2002) 518–544.
[6] A. Bar-Noy, R.E. Ladner, Windows scheduling problems for broadcast systems, SIAM Journal on Computing (SICOMP) 32 (4) (2003) 1091–1113.
[7] A. Bar-Noy, R.E. Ladner, T. Tamir, Windows scheduling as a restricted version of bin packing, ACM Transactions on Algorithms (TALG) 3 (3) (2007).
[8] A. Bar-Noy, J. Christensen, R.E. Ladner, T. Tamir, A general buffer scheme for the windows scheduling problem, in: Proc. of WEA, 2005.
[9] S.K. Baruah, N.K. Cohen, C.G. Plaxton, D.A. Varvel, Proportionate progress: A notion of fairness in resource allocation, Algorithmica 15 (6) (1996) 600–625.
[10] S.K. Baruah, S.-S. Lin, Pfair scheduling of generalized pinwheel task systems, Institute of Electrical and Electronics Engineers. Transactions on Computers 47 (1998) 812–816.
[11] J. Chung, J.W.S. Liu, K. Lin, Scheduling periodic jobs that allow imprecise results, Institute of Electrical and Electronics Engineers. Transactions on Computers 39 (9) (1990) 1156–1174.
[12] P.C. Fishburn, J.C. Lagarias, Pinwheel scheduling: Achievable densities, Algorithmica 34 (1) (2002) 14–38.
[13] R.M. Karp, A characterization of the minimum cycle mean in a digraph, Discrete Mathematics 23 (1978) 309–311.
[14] H.W. Lenstra, Integer programming with a fixed number of variables, Mathematics of Operations Research 8 (1983) 538–548.
[15] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM 20 (1) (1973) 46–61.
[16] M. Lindenbaum, Private communication.
[17] R. Tijdeman, The Chairman assignment Problem, Discrete Mathematics 32 (1980) 323–330.